AI Assistant Coding
Assignment - 2.4

Baisa Vaishnavi
2303A52142
Batch - 41

Task 1: Book Class Generation

❖ **Scenario:**

You are building a simple library management module.

❖ **Task:**

Use Cursor AI to generate a Python class Book with attributes title, author, and a summary() method.

Prompt:

Create a simple Python library management system. Generate a Book class with attributes title and author. Include a summary() method that returns book details.

Generated class :

C: > Users > Hello > Create a simple Python library management.py > ...

```python
#Create a simple Python library management system. Generate a Book class with attributes title and author.
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        return f"'{self.title}' by {self.author}"
# Example usage:
if __name__ == "__main__":
    book1 = Book("1984", "George Orwell")
    book2 = Book("To Kill a Mockingbird", "Harper Lee")

    print(book1.summary())
    print(book2.summary())
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Result: 57.0
PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14-64/python.exe
'1984' by George Orwell
'To Kill a Mockingbird' by Harper Lee
PS C:\Users\Hello>
```

Justification :
By finishing this task, we can display the book information as required in the question using Cursor AI. The code is easy to understand.

Task 2: Sorting Dictionaries with AI
❖ **Scenario:**
You need to sort user records by age.
❖ **Task:**
Use Gemini and Cursor AI to generate code that sorts a list of dictionaries by a key.

PROMPT :

CURSOR: Create efficient Python code to sort a list of dictionaries by the key "age". Each record contains "name" and "age"
GEMINI: #Generate Python code to sort a list of dictionaries containing user records by the key "age". Each dictionary has the keys: "name" and "age". Find and print names of users with the given age

Cursor:

5.py

```python
people = [
    {"name": "Alice", "age": 32},
    {"name": "Bob",   "age": 25},
    {"name": "Carol", "age": 29},
]

# Sort by age (ascending)
people_sorted = sorted(people, key=lambda person: person["age"])

print(people_sorted)
```

Output:

```
Problems  Output  Debug Console  Terminal  Ports                                   Python
[{'name': 'Bob', 'age': 25}, {'name': 'Carol', 'age': 29}, {'name': 'Alice', 'age': 32}]
PS C:\Users\Hello\aiassisstantcoding>
```

Gemini:

```python
people = [
    {"name": "Alice", "age": 30},
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35},
    {"name": "David", "age": 25} # Added a person with the same age to show stable sort
]

# Sort the list of dictionaries by the 'age' key
# The 'key' argument takes a function to extract a comparison key from each list element.
# lambda x: x['age'] creates an anonymous function that returns the 'age' value for each dictionary x.

sorted_people = sorted(people, key=lambda person: person['age'])

print("Original list:")
for person in people:
    print(person)

print("\nSorted list by age:")
for person in sorted_people:
    print(person)
```

Output:

```
Original list:
{'name': 'Alice', 'age': 30}
{'name': 'Bob', 'age': 25}
{'name': 'Charlie', 'age': 35}
{'name': 'David', 'age': 25}

Sorted list by age:
{'name': 'Bob', 'age': 25}
{'name': 'David', 'age': 25}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}
```

Justification:
In this task, even though we used two different AI tools, both produced the required output. we can clearly notice a difference in the way the code is written. The clarity of both codes is very good, but when it comes to output presentation, the output from Cursor AI is more neat and easier for anyone to understand.

Task 3: Calculator Using Functions
❖ **Scenario:**
You are reviewing a basic calculator module.
❖ **Task:**
Ask Gemini to generate a calculator using functions and explain how it

works.

PROMPT:
Write a python code for simple calculator with functions addition of n numbers, subtraction of n numbers, multiplication of n numbers, division of n numbers and power of n numbers which chooses the operation on the user input.

Gemini:

```python
def add_numbers(*args):
    """Returns the sum of all numbers."""
    return sum(args)

def subtract_numbers(*args):
    """Subtracts subsequent numbers from the first number."""
    if not args:
        return 0
    result = args[0]
    for num in args[1:]:
        result -= num
    return result

def multiply_numbers(*args):
    """Returns the product of all numbers."""
    if not args:
        return 1
    result = 1
    for num in args:
        result *= num
    return result

def divide_numbers(*args):
    """Divides the first number by subsequent numbers. Handles division by zero."""
    if not args:
        return 0 # Or raise an error, depending on desired behavior
    if len(args) == 1:
        return args[0] # Division by 1 is the number itself
    result = args[0]
    for num in args[1:]:
        if num == 0:
            print("Error: Cannot divide by zero!")
            return None # Indicate an error
        result /= num
    return result

def power_numbers(*args):
    """Calculates num1 ** num2 ** num3 ... (left-associative: (num1**num2)**num3)."""
    if len(args) < 2:
        print("Error: Power operation requires at least two numbers (base and exponent).")
        return None
```

```
    result = args[0]
    for num in args[1:]:
        result **= num
    return result

def get_numbers_input(prompt="Enter numbers separated by space: "):
    while True:
        try:
            nums_str = input(prompt)
            numbers = [float(x) for x in nums_str.split()]
            if not numbers:
                print("No numbers entered. Please try again.")
                continue
            return numbers
        except ValueError:
            print("Invalid input. Please enter numbers only, separated by spaces.")

def calculator():
    print("\n--- Simple Calculator ---")
    while True:
        print("\nSelect operation:")
        print("1. Add (n numbers)")
        print("2. Subtract (n numbers)")
        print("3. Multiply (n numbers)")
        print("4. Divide (n numbers)")
        print("5. Power (n numbers: (num1**num2)**num3...)")
        print("6. Exit")
```

```
        choice = input("Enter choice(1/2/3/4/5/6): ")

        if choice in ('1', '2', '3', '4', '5'):
            numbers = get_numbers_input()
            if numbers is None: # If get_numbers_input had an issue (e.g., empty input)
                continue

            result = None
            if choice == '1':
                result = add_numbers(*numbers)
            elif choice == '2':
                result = subtract_numbers(*numbers)
            elif choice == '3':
                result = multiply_numbers(*numbers)
            elif choice == '4':
                result = divide_numbers(*numbers)
            elif choice == '5':
                result = power_numbers(*numbers)

            if result is not None:
                print(f"Result: {result}")
        elif choice == '6':
            print("Exiting calculator. Goodbye!")
            break
        else:
            print("Invalid input. Please enter a valid choice.")

# Run the calculator
calculator()
```

Output:

```
--- Simple Calculator ---

Select operation:
1. Add (n numbers)
2. Subtract (n numbers)
3. Multiply (n numbers)
4. Divide (n numbers)
5. Power (n numbers: (num1**num2)**num3...)
6. Exit
Enter choice(1/2/3/4/5/6):
```

Vs Code:

Welcome | #check whether the given number is Armst Untitled-1 | Write a python code for simple calculator.py

C: > Users > Hello > Write a python code for simple calculator.py > ...

```python
#Write a python code for simple calculator with functions addition of n numbers, subtraction of
def add(numbers):
    return sum(numbers)

def subtract(numbers):
    result = numbers[0]
    for num in numbers[1:]:
        result -= num
    return result

def multiply(numbers):
    result = 1
    for num in numbers:
        result *= num
    return result

def divide(numbers):
    result = numbers[0]
    for num in numbers[1:]:
        if num == 0:
            return "Error: Division by zero"
        result /= num
    return result

def power(numbers):
    base = numbers[0]
    exponent = numbers[1]
    return base ** exponent

def calculator():
    print("Simple Calculator")
    print("Operations:")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Power")
```

C: > Users > Hello > Write a python code for simple calculator.py > ...

```
30  def calculator():
38
39      choice = input("Enter operation number (1-5): ")
40
41      if choice in ['1', '2', '3', '4', '5']:
42          n = int(input("Enter the number of operands: "))
43          numbers = []
44          for i in range(n):
45              num = float(input(f"Enter operand {i+1}: "))
46              numbers.append(num)
47
48          if choice == '1':
49              print(f"Result: {add(numbers)}")
50          elif choice == '2':
51              print(f"Result: {subtract(numbers)}")
52          elif choice == '3':
53              print(f"Result: {multiply(numbers)}")
54          elif choice == '4':
55              print(f"Result: {divide(numbers)}")
56          elif choice == '5':
57              if len(numbers) != 2:
58                  print("Error: Power operation requires exactly two operands.")
59              else:
60                  print(f"Result: {power(numbers)}")
61      else:
62          print("Invalid operation number.")
63
64  calculator()
```

Output :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Hello> & C:/Users/Hello/AppData/Local/Python/pythoncore-3.14-64/python.exe
Simple Calculator
Operations:
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Power
Enter operation number (1-5): 1
Enter the number of operands: 2
Enter operand 1: 23
Enter operand 2: 34
Result: 57.0
PS C:\Users\Hello>
```

Justification:

By finishing this task the simple basic calculator, The program is easy to understand because each operation is written in a separate function. It can perform calculations on more than two

numbers, so it is more useful than a normal calculator. The program checks for errors like wrong input and division by zero, so it runs safely.

Task 4: Armstrong Number Optimization

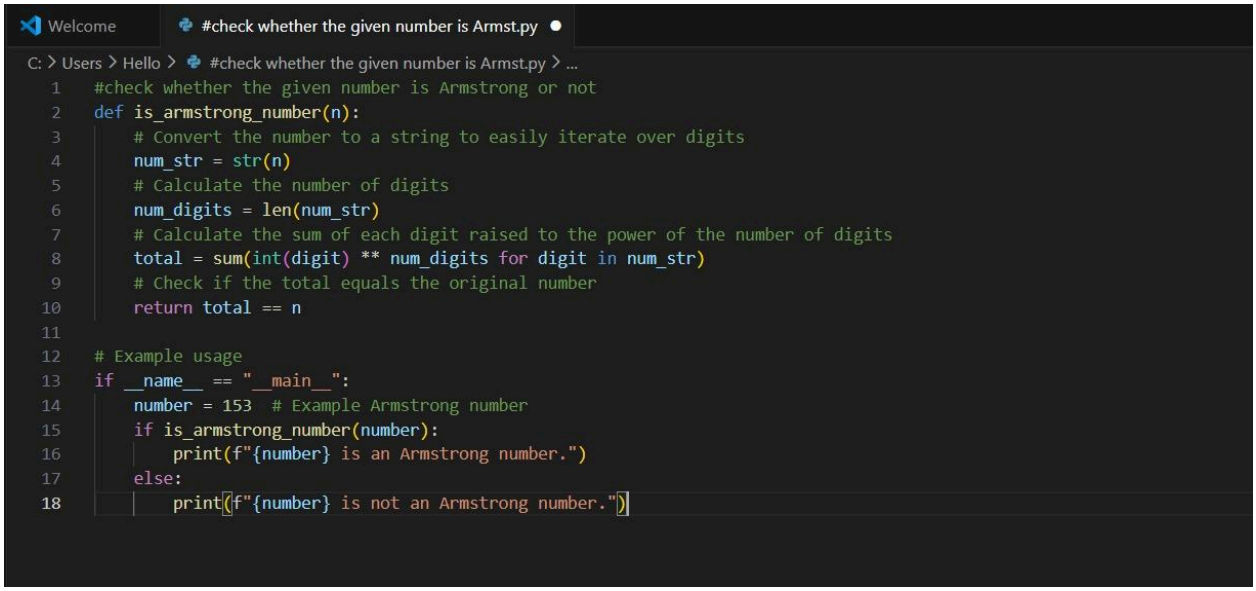
❖ Scenario:

An existing solution is inefficient.

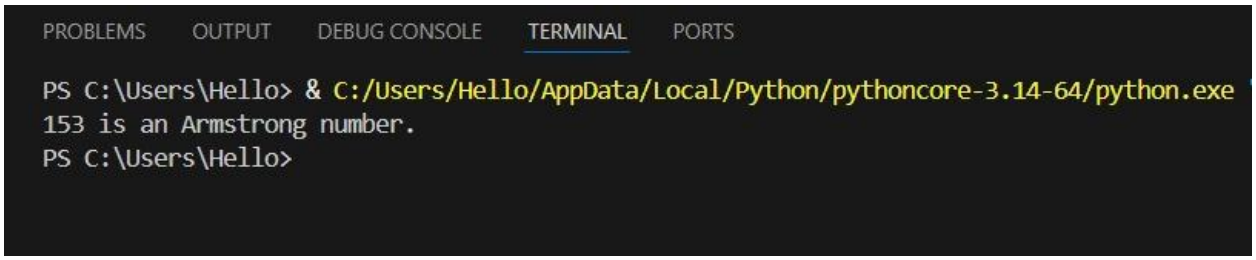
❖ Task:

Generate an Armstrong number program using Gemini, then improve it using Cursor AI.

Gemini :



Output:

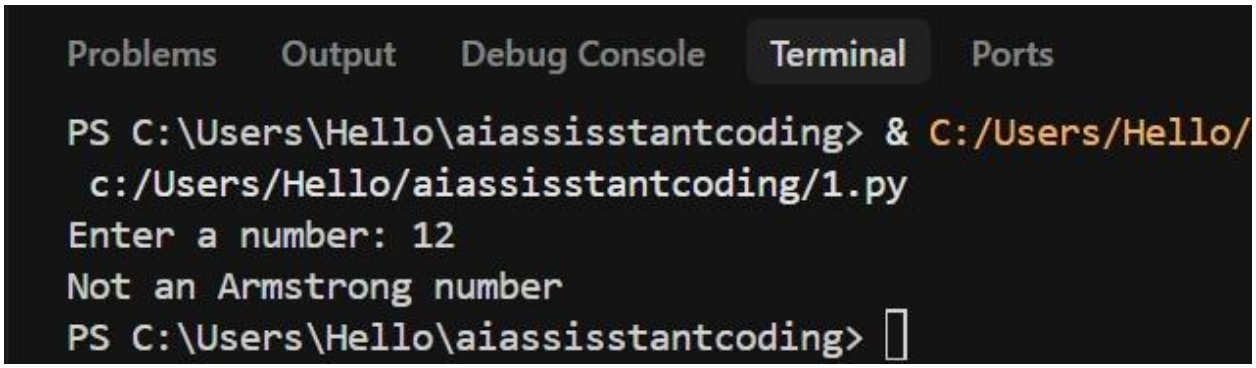


Cursor AI :

1.py

```python
# Optimized and simplified Armstrong number checker

num = int(input("Enter a number: "))

# Convert number to string to determine the number of digits (O(1) for conversion)
digits = len(str(num))

# Use a generator expression to calculate the Armstrong sum in a single pass
armstrong_sum = sum(int(d)**digits for d in str(num))

if armstrong_sum == num:
    print("Armstrong number")
else:
    print("Not an Armstrong number")

# Improvements:
# - Uses string conversion to get digit count efficiently.
# - Single pass to both access digits and raise them to power.
# - Generator expression for readability and performance.
# - Removed unnecessary variables and duplicated code.
```

Output :



Justification:

In this task, we can clearly observe that, the code given by gemini is a bit complex and has more time complexity which makes code less efficient, as we have optimized it in cursor ai, the code is optimized version with only basic syntax which makes the code easy to understand for anyone. Even though the outputs are same, both the codes given by both gemini and cursor ai are efficient.